Mining the EAGF's Application Process of EU Direct Payments for Farmers

Abdel K. Bokharouss, Stef Creemers, Casper van der Schenk, Jelmer Wilhelm

Eindhoven University of Technology

October 1, 2018

Abstract

Several processes are analysed in relation to- and in the context of- the Business Process Intelligence Challenge 2018. The processes that are analysed, describe the application processes for EU direct payments from the European Agricultural Guarantee Fund (EAGF). These applications are, for example, for subsidies that are facilitated to farmers. This guarantees a lower bound income for the aforementioned farmers that is decoupled from the production levels that is achieved by them. These processes are, in relation to a set of business questions, analysed and described using various process- and data-mining techniques and tools.

1 Introduction

This report encapsulates our findings in regard to the Business Process Intelligence Challenge (BPIC) 2018. In this challenge, sponsored by Celonis, NWO's DeLiBiDa project and Minit, the various partners provided us with a real-life event log and subsequently presented several (analytical) business questions. The objective of this report is to summarize the tools, methods and techniques that were used to answer these questions.

The European Union (EU) aims to keep its agricultural sector competitive, sustainable and ecological. The EU views the agricultural sector as one of the key drivers for sustainable economic development in the region. To assist the incentives and innovation in this sector, it spends a reasonable amount of its budget on income support for farmers, market measures and rural development pro grammes that are encapsulated in its Common Agricultural Policy (CAP). The processes that are considered in the context of the Business Process Intelligence Challenge 2018, are applications for EU direct payments from the European Agricultural Guarantee Fund (EAGF), that are facilitated to for example farmers, provide a guaranteed lower bound income for the aforementioned farmers that is decoupled from the production levels that are achieved. This application process, repeats every year with a few minor (administrative) changes that result from adjustments in the (agricultural) regulations of the European Union. These applications are followed by rigorous onsite (e.g. land of the farmers) inspections in around one tenth of the cases.

The data [VDB18] in this challenge is provided by the German company *data experts*, located in Neubrandenberg. The provided data is from their Java Enterprise system *profil c/s* [de]. In particular, the data that was provided was in relation to processes at the level of federal ministries of agriculture and local departments. The system, *profil c/s*, supports various kinds of administrative processes. The data used in the challenge is, however, focused on one particular flow of events. As was explained before, The challenge is centered around the yearly allocation of direct payments.

The workflows in *profil c/s* are implied by the various document types that come into play in the overlaying process, as will be explained in more detail in *2.The Processes*. Each document type and associated state, has a set of actions accredit to it. The set of actions can be scheduled automatically, but its is also possible that these actions are manually executed. This can happen at any point in time (i.e. state) in the process. If automatically executed actions are not explicitly label as such, it is possible to infer it from peculiarities such as a high number of actions that are executed by the same user in a negligible amount of time (i.e. batch-processing).

The remainder of the report is structured as follows. Section *2.The Processes* covers the data itself. It provides a description of the different attributes and documents found in the data, as well as introducing the main process model with a description of different sub processes. Then, the data is preprocessed as described in section *3. Preprocessing and exploration of the event logs*. In section *4. Undesired Outcomes*, we look at several undesired outcomes and try to predict them using application attributes and other derived attributes. The same is done in section *5. Risk Assessment: Prediction of Penalties*, trying to predict penalties. Then, in section *6. Differences between Departments and across Years*, differences between departments and years are described. Lastly, in section *7. Discussion* a discussion is provided.

2 The General Process

In this section, the data that was provided by the European Agricultural Guarantee Fund will be explained in terms of the concepts that are used. The explanation of the attributes and the types of documents are stated in the first part. In the second part an overview of the process will be given by means of a BPMN model to show the main generalized process. On the basis of this model the generalized way of how a case goes trough the process will be explained.

2.1 Data description

The BPI Challenge of 2018 provided an event log that covers the handling of the applications for EU payments for German farmers from the European Agricultural Guarantee Fund as mentioned earlier. It consists of 43,809 applications with a total of 2,514,266 events over a period of 3 years. The applications are handled by four different departments called 4e, 6b, 4d and e7 which handle respectively 31, 25, 13, 30 percent of all the applications. For each application the following data is known:

- `program-id` - Internal id of the funding program
- `concept:name` - Unique case id for the application
- `identity:id` - Globally unique case id (UUID)
- `Department` - Id of the local department
- `application` - The applicant's id, the same across years
- `year` - The current year
- `number of parcels` - The number of parcels
- `area` - The total area of all parcels
- `basic_payment` - Application for basic payment scheme
- `greening` - Application for greening payment
- `redistribution` - Application for re-distributive payment
- `smaller farmer` - Application for small farmer scheme

- `young farmer` - Application for payment for young farmers
- `applicant` - Anonymized identifier of applicants

Furthermore, there are eight documents made per case which are used in different moments in the process. Each document has its own sub-process. There is one leading document for which the other documents are created. The leading document is the application document. For each created document the sub-process consists in most cases of an initialize action, followed by a editing action which will be ended with a finish editing action. During the sub-process extra information can be recorded. Below you find the different types of documents:

- **Control summary** - A document containing the summarized results of various checks (reference alignment, department control, inspections)
- **Department control parcels (before 2017** - A document containing the results of checks regarding the validity of parcels of a single applicant
- **Entitlement application** - The application document for entitlements, i.e., the right to apply for direct payments, usually created once at the beginning of a new funding period
- **Inspection** - A document containing the results of on-site or remote-inspections
- **Parcel Document (before 2016)** - The document containing all parcels for which subsidies are requested
- **Geo Parcel Document (replaces Parcel document since 2016 and Department control parcels since 2017)** - The document containing all parcels for which subsidies are requested. From 2017, the Geo Parcel Document also replaces the Department control parcels document.
- **Payment application** - The application document for direct payments, usually each year
- **Reference alignment** - A document containing the results of aligning the parcels as stated by the applicant with known reference parcels (e.g. a cadaster)

2.2 Process description

The first step to get more insights into the process was to see how cases flow through the process and to see if they follow a certain main process. To do this we used ProM and Disco which gave a broad overview off the process. As every case has a lot of events, we minimize paths and activities to learn about the most common events. These are used as baseline for a generalized model off the application process. The idea of a generalized model is to get a clear insight into the process. Because the Parcel Document and the Department control parcels are replaced by the Geo Parcel Document in different years and those documents are before 2017 part of the main process, the first generalized model does not conform with the dataset. Therefore, the choice is made to make a generalized process model for 2017 as this is the most recent process and therefore the most useful to compare with for readers. Moreover, it allows for one standard model to compare other models with. Celonis is used to enrich the model with important events until the right main process was found. This process is found in Figure 1 and will be used to describe the main process for 2017 in eight steps.

1. Sub-procces - A new payment application comes in by mail. When it is a seemingly credible application it will be accepted as valid. The 'Payment Application' document is created as main document for the application.
2. Sub-procces - The 'Geo Parcel' document is made to register which subsidies are requested for all mentioned parcels. This also includes the 'Parcel document' which was replaced in 2016 and was a document containing all parcels for which subsidies are requested. More on this will be explained in section 6.

3. Sub-procces - The document ‘Inspection’ contains results of on site or remote inspections if there were any (this happens in about seven percent of the cases).

4. sub-procces - The summary of all the activities and results of checks are added to the ‘Control Summary’ document.

5. sub-procces - The alignment document is made to contain the results of aligning the parcels as stated by the applicant and added to the ‘Payment Application’.

6. sub-procces - The ‘Payment Application’ document is taken further into the progress and the department will start calculating the amount it will grant.

7. sub-procces - Before the calculations start the Geo Parcel document will again be checked regarding the validity of parcels of a single applicant. The ‘Department Control Parcels’ did this before 2017. More on this will be explained in section 6.

8. sub-procces - The last step of the process consist of official starting the payment, or abort the payment if something went wrong. Then changes have to be made and after that the payment can officially start.

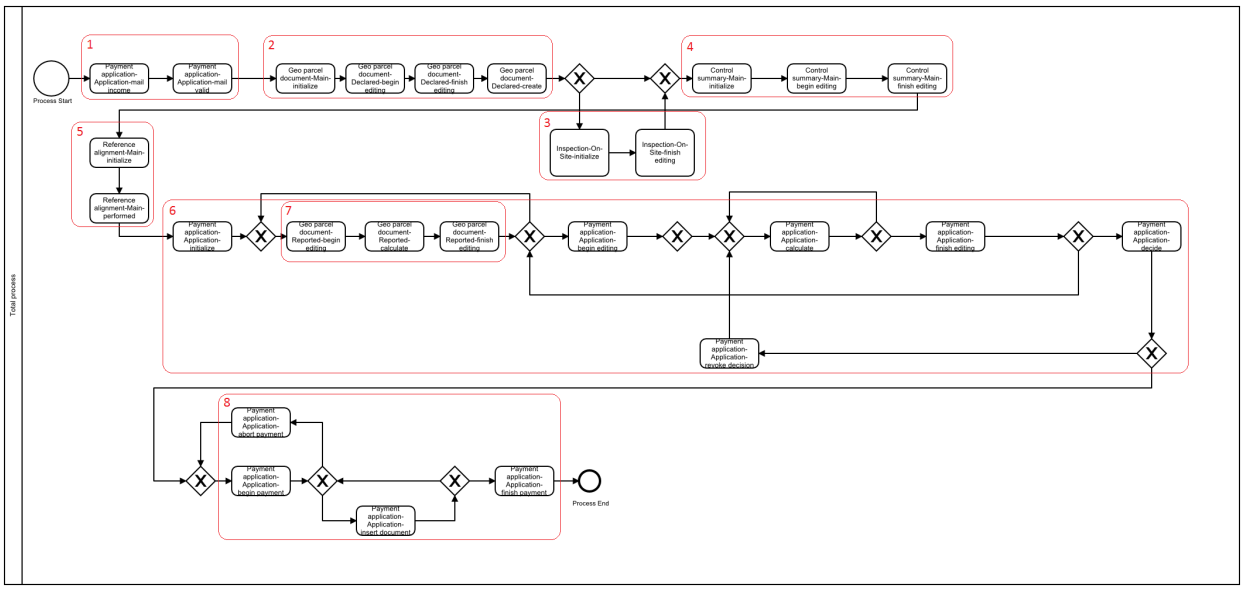


Figure 1: Generalized model 2017

To ensure proper understanding of the main process PI conformance in Celonis was used. This makes it possible to compare a model with the reality and it will show all the violations that the model has. For this model is 21% PI conformance reached. This sounds not as a lot, however this model is a generalized model of 2017 and every deviation of the main process will result in a decrease of the PI conformance, besides an implication of the PI conformance is that it has a hard time to see the event order if they occur on small time intervals. For example, some events are processed on exactly the same timestamps. Celonis will differentiate between the two event orders (while they occur at the same time) instead of adapting one as standard. As such, it sometimes marks cases with the same order of events as the BPMN model as non-conforming. These things taken into account, when looking through all the violations gave confidence that this model shows the main process (including the inspections) of the dataset for 2017.

3 Preprocessing and exploration of the event logs

In this section, the event logs (i.e. the data) will be observed, processed and extended if applicable. In particular, this section focuses on the overall preprocessing operations that were applied to the data, and the logic, assumptions and observations that support the (need for) these steps.
In particular, the data is going to be sanitized and 'reshaped' to, for example, forms that are more appropriate for tasks that involve predictive analytics as is the case in, for example, sections *4. Undesired Outcomes* and *5. Risk Assessment: Prediction of Penalties*. These sections will describe, if needed, more in-depth preprocessing steps to answer the particular business questions.

3.1 Formation of attributes

Once the shape completeness and overall characteristics were analyzed, the most straightforward data sanitization step that was deemed necessary was the uniformness of `NA` values in the dataset (i.e. the event log). The data contains, namely, different value-types and/or conventions to indicate undefined, unrepresentable and/or missing data in the event log. One clear example is the `Resource` attribute. It contains the string values '`0;n/a`'. There are many reasons to strive for one unique convention (e.g. an empty entry) to indicate `NA` values across the dataset. In this particular case, we are dealing with a categorical attribute (`Resource`). Carelessness in some future analytics steps could result in the ignorance of the existence of this value and thus the inclusion of a one more category, even though it seems to be actually missing data. The term 'seems' is used since it can actually be the case in some situation that `NA` values hold information in a particular context.

After a brief analysis of the range of values and an attempt to find a value for these `NA` values in the `Resource` attribute through techniques related to Association Rule Mining, the choice was made to simply replace these values by the convention that was chosen to be used in this research project: an empty string. In particular, all the different types of `NA` values such as '`0;n/a`' are replaced by a special floating point value: `np.nan` [Gan04] (see snippet *Code snippet 1* below). It is defined in, for example, the context of the *Numpy* package (e.g. for scientific computing) for the language that is Python, which is next to *R*, the go-to language for scripting tasks in this research project. When converted to data-files such as comma-separated values files, the entries simply show as empty entries which is often times the convention used in other tooling for `NA` values.

```
for sp in all_proc:
    sp.loc[sp['Resource'] == '0;n/a', 'Resource'] = np.NaN
```

Code snippet 1: Where `sp` is a placeholder for subprocesses (e.g. reference alignment, payment application etc.) and `all_proc` is a list of the event logs of all the processes.

The next step in the general preprocessing step is related to the various identifiers in the event logs. Most of the identifiers, such as `(case) applicant` and `(case) application`, seem to be formatted in a hexadecimal notation. However, other identifiers such as `Case ID`, `docid` and `eventid` seem to be formatted as plain (negative) integers. The choice is, therefore, made to process these identifiers to a hexadecimal notation like the aforementioned identifiers. A snippet of the python code is devised in *Code snippet 2* below.

```
sp['Case_ID'] = sp['Case_ID'].apply(
        lambda x: x if np.isnan(x) else bitstring.BitArray('int:64=%d' % x).hex)
```

Code snippet 2: Where `sp` is a placeholder for subprocesses (e.g. reference alignment, payment application)

This makes the identifier better readable and recognizable. In addition, this step was done with the assumption that the negative identifiers were formatted that way due to a mistake and/or faulty data processing steps. It seemed odd to have a set of identifiers in hexadecimal notation and use negative integers as identifiers for another set of identifiers.

3.2 Redundant information

Some information in the event logs seems to be redundant. In the event log we see, for example, that the `Case ID` and `(case) application` are identical attributes. The latter one is, therefore, dropped. Other attributes, such as `Variant` and `Variant index` are dropped since they do not proper documentation and do not seem to entail a lot of information considering the various range of values they hold. Another set of attributes that is dropped is `(case) program-id` and `lifecycle:transition` which have only one unique value across their respective columns. Their information entropy in this analysis is, therefore, zero (i.e. has no value).The attributes `activity` and `concept:name` are also the same and the latter one is therefore dropped.

And lastly it was observed that the `Activity` attribute is the composition of the `doctype`, `subprocess` and `activity` attributes (*Code Snippet 3* evaluates to `True`), but all of these attributes were kept despite the introduction of redundant information since it eases analytical tasks in some situations.

```
elog['Activity'].equals(
        elog['doctype'] + '-' + elog['subprocess'] + '-' + elog['activity'])
```

Code snippet 3: Statement to confirm the aformentioned hypothesis.

3.3 Feature engineering

The next step is to analyze the large set of attributes that are facilitated in the event logs and their respective information value. The representation of the information encapsulated in an attribute can often times not directly be used in, for example, predictive modeling. Feature engineering is often times a necessary intermediate step to get all the information one can extract- and subsequently exploit from an attribute.

The first (set of) attributes that are going to be further analyzed and subsequently processed are the attributes `(case) amount_appliedX` where `X` $\in \{0, 1, 2, 3\}$. The applied amount is trivially the amount that was requested by the applicant. The numerical suffix is an indication of the current payment process, starting with zero. A new payment process can be initiated when a case requires changes due to, for example, objections by the applicant and/or requests of the departments involved. The choice was made to engineer several attributes from this set of attributes. In particular, the following attributes were devised:

- `num changes applied amount` - The number of times the applied amount has changed in the entire process.
- `final applied amount` - The final applied amount
  (i.e. arg $max_{\texttt{x} \in \{0,1,2,3\}}$ `(case) amount_appliedX` $!=$ `NA`)
- `diff applied amount` - The difference between the first noted applied amount and `final applied amount`.
- `applied amount lowered` - Encapsulates whether the applied amount has been lowered over the entire process (`boolean` value).
- `applied amount changed` - Encapsulates whether the applied amount has been changed at all over the entire process (`boolean` value).

These attributes were calculated using various function. As an example, a code snippet for `final applied amount` is devised below in *Code Snippet 4*.

```
def return_final_applied_amount(x):
    if not pd.isnull(x['(case) amount_applied3']):
        return x['(case) amount_applied3']
    elif not pd.isnull(x['(case) amount_applied2']):
        return x['(case) amount_applied2']
    elif not pd.isnull(x['(case) amount_applied1']):
        return x['(case) amount_applied1']
    else:
        return x['(case) amount_applied0']
```

Code snippet 4: Function used for the calculation of the final applied amount.

Experimental evaluations has to show whether the construction of these attributes is actually useful. The next (set of) attributes that is going to be processed is rather similar to the applied amount attributes. The set of attributes `(case) payment_actualX` where `X` $\in \{0, 1, 2, 3\}$ is going to be processed in almost the same fashion, but there is one major difference with the feature engineering process of the applied amount attributes.

Complete Timestamp	(case) amount_applied0	(case) amount_applied1	(case) amount_applied2	(case) amount_applied3
2017-04-25 14:39:54.939	587772.08	587472.31	587373.98	587261.11
2017-06-21 16:38:56.677	56564.75	56564.75	56482.47	56550.60
2016-11-30 12:58:19.228	34769.12	34794.71	35254.05	35253.72

(a) `The amount_appliedX attributes`

(case) payment_actual0	(case) payment_actual1	(case) payment_actual2	(case) payment_actual3
54786.25	0.00	-252.02	0.0
42266.33	0.00	-1448.64	0.0
92420.70	-546.08	-3077.42	0.0

(b) The `payment_actualX` attributes

Figure 2: Difference between the `amount_appliedX` and `payment_actualX` attributes.

The changes in the applied amounts seem to be absolute changes, while the changes in the actual payments amounts seem to be relative changes, as can be seen in Figure 2. In other words, the changes in the actual payments attributes seem to be of a corrective nature. This difference might seem subtle, but it would have major implications for the validity of the featured attributes if it was not spotted. Building on that, the following attributes were devised in relation to the `payment_actualX` attributes:

- `num changes actual amount` - The number of times the actual amount has changed in the entire process.
- `final actual amount` - The final actual amount (sum over all the non-null `payment_actual` attributes: $\sum_{\texttt{X} \in \{0,1,2,3\} \wedge \texttt{(case) payment\_actualX != NA}} \texttt{(case) payment\_actualX}$)
- `diff actual amount` - The difference between the first noted actual amount and `final actual amount`.
- `actual amount lowered` - Encapsulates whether the actual amount has been lowered over the entire process (`boolean` value).
- `actual amount changed` - Encapsulates whether the applied amount has been changed at all over the entire process (`boolean` value).

The last set of attributes that is processed in more or less the same manner as the is the set of attributes related to the penalties. In particular, the set of attributes `(case) penalty_amountX` where `X` $\in \{0, 1, 2, 3\}$. are used to devise the following set of attributes:

- `num changes penalty amount` - The number of times the penalty amount has changed in the lifecycle of the particular case.
- `final penalty amount` - The final penalty amount (i.e. arg $max_{\texttt{x} \in \{0,1,2,3\}}$ `(case) penalty_amountX` $!=$ `NA`)

In addition, more in depth penalty classification was applied. In particular, the number of penalties and the number of severe penalties were counted and encapsulated in the following attributes:

- `num penalty reasons` - The number of different penalties that were applied for this case. This is calculated by considering the number of boolean attributes of the form `(case) penalty_X` that evaluate to `true`.
- `num severe penalties` - The number of severe penalties that were applied. Severe penalties is any of the penalties in {B3, B4, B5, B5F, B6, B16, BGK, C16, JLP3, V5}.

This concludes the general preprocessing section. Additional preprocessing steps will be described in other sections such as *4.2 More specific preprocessing*. This necessity for these additional steps depends on the particular business question being answered and the choice of process- and/or data-mining techniques that are chosen to be exploited.

4 Undesired Outcomes

The first business question that is going to be tackled in this report is related to 'Undesired outcomes' in the application process that is devised in *2. The Process*. A case that follows the general flow of events starts it life cycle around May of the respective year and *should* be closed no later than the end of that particular year. In this context, 'closed' refers to a timely payment of the to be granted subsidies. Some cases deviate from this flow of event and respective time-line. In particular, each year there are some cases that are classified by the term *Undesired outcome.*. These cases can be divided into two classes:

Undesired outcome #1 is used to indicate a case in which the payment is late. On retrospect, a payment is considered to be timely if the particular case in question has `begin payment` activity by the end of the year that was not followed by an `abort payment` activity.

Undesired outcome #2 is used to indicate a case that needs to be reopened. This can be initiated by two (set of) parties:

- *the department*, which can be inferred by the subprocess `Change`
- due to legal objection(s) by *the applicant* (subprocess `Objection`)

This can result in additional payments and/or reimbursements. To assess whether this is actually the case, one can look at the `payment_actual{x}` cases where $x > 0$ (i.e. the xth payment after the initial one).

The question is whether it is possible to detect these cases as early as possible. Ideally, this would happen before a decision is made for the case in question, which is indicated by the first occurrence of the activity `Payment application-Application-decide` (i.e. *doctype* = `Payment Application`, *subprocess* = `Application` and *activity* = `decide`) . One is allowed to use data of previous year to answer this question.

4.1 Approach

There are various ways to assess whether it is possible to detect such cases. The most sensible thing to do is to use predictive modeling to answer this question. The objective is thus to predict whether a case will end being one of the two types of undesired outcomes, at and/or before a particular stage or point in time in the life-cycle of the case. To perform this predictive modeling, more preprocessing steps are required as will be elaborated on in *4.2 More specific preprocessing*. Depending on the performance metrics that are measured once the models are built and trained, we can answer the question in a scientifically sound manner.

4.2 More specific preprocessing

The data has already been formatted, sanitized and enriched as describe in section *3. Preprocessing and exploration of the event logs*. However, additional preprocessing steps are needed considering the approach that was described in *4.1 Approach*. In particular, we need to reshape and extend the dataset at hand in such a way that predictive modeling is actually possible.

First and foremost, a choice has to be made whether one still wants to consider multiple records for the same case. A better way of reshaping the data for predictive modeling is to have, for example, one record per case in which all the data is encapsulated needed for the prediction, extended with the relevant target attributes. This is, therefore, the approach that will be followed. Recall that one wants to detect the cases as early as possible. Ideally, this would happen before a decision is made for the case in question, which is indicated by the first occurrence of the activity `Payment application-Application-decide`. The choice is, therefore, made to consider the following activity that always occurs just before the aforementioned activity: `Payment application-Application-finish editing`. By doing so, we are left with a dataset of 43793 records (and thus unique cases). Only 16 cases needed to be removed due to absences of the `Payment application-Application-decide` activity in their life-cycle.

The next step would be the generation of the target attributes since they are only implicitly present in the datasets. The choice is made to construct three target (dichotomous) attributes: `undesired_outcome1`, `undesired_outcome2` and `undesired_outcome`. The definition two attributes should be rather trivial. The latter attribute is the just a logical or of the first two attributes. In other words, it entails whether either of the two undesired outcomes are registered for a case. The objective is to predict exactly which undesired outcome is applicable, but if the performance metrics of these models are poor, a shift towards models that try to predict whether either of the undesired outcomes (i.e. `undesired_outcome` as target attribute) will be made since this should be intuitively easier to learn than more specific models.

The following two functions were used for the generation of the two target attributes `undesired_outcome1` and `undesired_outcome2`:

```
def is_undesired_outcome1(x):
    case_id = x['Case ID']
    case_subset = elog[elog['Case ID'] == case_id]
    case_events = set(case_subset['Activity'])
    if 'Payment application-Application-begin payment' in case_events:
        if 'Payment application-Application-abort payment' not in case_events:
            return True # no abort payment activity
        else: # check whether it was before the end of the year
            earliest_date = min(list(case_subset['timestamp_year']))
            date_abort_payment = list(case_subset[case_subset['Activity'] ==
                                      'Payment application-Application-abort payment']
                                      ['timestamp_year'])[0]
            if date_abort_payment > earliest_date:
                return True # 'abort payment' was not before the end of the year
    return False # no begin payment activity
```

Code snippet 5: Function that is used for the calculation of `undesired_outcome1`

Recall that the first type of undesired outcome indicates a case in which the payment is late. And on retrospect, a payment is considered to be timely if the particular case in question has `begin payment` activity by the end of the year that was not followed by an `abort payment` activity. The function depicted in *Code Snippet 5*, therefore, checks whether the event `Payment application-Application-begin payment` is in the life-cycle of the case (as far as can be judged from the data) and if that is the case it checks for the presence of the `Payment application-Application-abort payment` in its life-cycle. If it is present, it subsequently checks whether it happened before the end of the year.

```
def is_undesired_outcome2(x):
    case_id = x['Case ID']
    case_subset = elog[elog['Case ID'] == case_id]
    case_subprocesses = set(case_subset['subprocess'])
    if 'Change' in case_subprocesses or 'Objection' in case_subprocesses:
        return True
    else:
        return False
```

Code snippet 6: Function that is used for the calculation of `undesired_outcome2`

The evaluation of the second type of undesired outcomes is a little bit less computational heavy. Recall that the second type of undesired outcomes is used to indicate a case that needs to be reopened. This can be initiated by two (set of) parties: 1) *the department*, which can be inferred by the subprocess `Change` and 2) due to legal objection(s) by *the applicant* (subprocess `Objection`). This is exactly what the function in *Code Snippet 6* checks for. It simply checks whether these subprocess are present in the life-cycle of the particular cases.

The subsequent calls are depicted in *Code Snippet 7*. The dataset `elog_uo` is the reshaped dataset that is being prepared for predictive modeling for undesired outcomes while `elog` is just the original complete event log after the preprocessing steps that are described in *3 Preprocessing and exploration of the event logs*. The complete event log `elog` is still needed since we need to have the access to the complete life-cycles of the cases to identify the two different types of undesired outcomes.

```
elog_uo['undesired_outcome1'] = elog_uo.apply(is_undesired_outcome1, axis=1)
elog_uo['undesired_outcome2'] = elog_uo.apply(is_undesired_outcome2, axis=1)
elog_uo['undesired_outcome'] = np.where(elog_uo['undesired_outcome1'] ||
                                        elog_uo['undesired_outcome2'], True, False)
```

Code snippet 7: The subsequent function calls.

4.3 Predictive Modeling: Detecting Undesired Outcomes

And now, after two rounds of extensive preprocessing: an elaborate overview of the approach, techniques and libraries that are used for the the detection of undesired outcomes.
The dataset that has been prepared for the task of predicting (i.e. detecting) undesired outcomes contains more attributes than initially was the case. While it has been extended with a lot of attributes that extract a lot of (implicit) information out of (combinations of) other attributes, choosing what set of attributes to use in the task of predictive modeling has also become substantially harder. While a significant number of attributes are not usable for the task of predictive modeling because of their nature, simply exploiting a trial-and-error based approach in the selection of attributes and subsequently building a model based on them is deemed to be an inefficient and non-optimal approach. In addition, there are a lot of machine learning models and principles that can be used in the problem context.

The choice is, therefore, made to take full advantage of a set of tools and concepts that recently came to light. An example of one of these recent developments is the release of a paper called 'ATM: A distributed, collaborative, scalable system for automated machine learning,' which was presented during the IEEE International Conference on Big Data in December 2017 [SDC+17]. At the aforementioned conference, researchers from MIT and Michigan State University presented a new system that automates the model selection step. Its performance was often times better than some experienced and knowledgeable Data Scientists. To quote the researchers and their non-technical marketing description of the system:

> *"Data scientists must shepherd their raw data through a complex series of steps, each one requiring many human-driven decisions. The last step in the process, deciding on a modeling technique, is particularly crucial. There are hundreds of techniques to choose from — from neural networks to support vector machines — and selecting the best one can mean millions of dollars of additional revenue, or the difference between spotting a flaw in critical medical devices and missing it."*

As was already described in the introduction of this paragraph, these new systems, research findings and other developed tooling can make it easier to identify which modeling technique is the best suited for the job at hand. In line with these recent breakthroughs in academia and in the industry, RapidMiner introduced a new addition to its already popular platform. The Auto Model feature was namely introduced in version 8.1. This is a feature that shares a lot of similarities with the aforementioned system that was introduced by MIT— and Michigan State University researchers.

The choice is, therefore, made to exploit this new feature to identify which modeling technique seems to be the best choice for the problem and data at hand. Once a (subset) of modeling techniques has been identified, the (subset of) model(s) are going to be rebuild and tweaked in Python. This allows one to have more control over the built model and it facilitates reproducibility of the results when one releases his/her source code.

And since RapidMiner is going to be used, the choice is also made to use its 'Feature Selection' attribute to identify the most useful predictive attributes in the pool of attributes that are present in the dataset and the derived attributes that were introduced in section *3. Preprocessing and exploration of the event logs*. Various wrapper methods such as forward selection, backward elimination and selection optimization methods (e.g. brute force, evolutionary approach) have been explored and evaluated to identify and select the most relevant attributes for the predictive modeling task at hand. These methods consider all possible subsets of features in their underlying implementations. There is also a lot of room for modifiability in these methods. In the evolutionary approach for example, the choice was made to use Random-Forest and Cross Validation internally in the Optimize Selection (Evolutionary) process.

The features that were identified to have the highest correlation with the occurrence of an undesired outcome are: `(case) amount_applied0`, `(case) area`, `(case) department`, `(case) number_parcels`, `(case) payment_actual0`, `(case) penalty_amount0`, `(case) penalty_B2`, `(case) year`, `final applied amount`, `final actual amount`, `final penalty amount`, `num_penalty_reasons`, `num_severe_penalties`, `num changes actual amount`, `num changes applied amount`, `num changes penalty amount`, `penalty amount applied`, `penalty applied`, `severe penalty applied`, `timestamp_month`, `timestamp_year`.
Considering the results of this selection, one can conclude without a doubt that the preprocessing and feature engineering efforts that are described in section hyperref[sec:Preprocessing and exploration of the event logs]*3. Preprocessing and exploration of the event logs* were worth the investment and effort considering the relatively high proportion of feature engineering attributes among the listed set of attributes.

The choice was made to start with `undesired_outcome` as target attribute since the results should intuitively be the best for this attribute. This attribute has, coincidentally almost a perfect 50/50 split (49.8/50.2). The modeling techniques Naive Bayes, Deep Learning, Decision Tree, Random Forest and Gradient Boosted Trees were selected in the aforementioned Auto-model feature of RapidMiner. The expectation for the first selected model were not high, but having a baseline is never unpleasant in a research setting. The results exceeded the initial expectations. The accuracies of the different models were 57%, 90%, 97%, 96% and 97% respectively. The other performance metrics were also positive. Consider, for example, the confusion matrix (Table 1) for the Deep Learning model below.

		Predicted Value		
		True	False	Precision
Actual Value	True	3801	293	92.85%
	False	567	4086	87.65%
	Recall	86.8%	93.3%	

Table 1: The confusion matrix that resulted from the Deep Learning Model.

After the evaluation of the results of these models, an attempt was made to build models for the other two target attributes that were derived in *4.2 More specific preprocessing*: `undesired_outcome1` and `undesired_outcome2`. The latter one has a rather 'skewed' distribution in the processed dataset (i.e. `False` occurs way more often), as can be seen in *Figure 3*. Note that only 20% of the records are used in the performance evaluation. This is, therefore, dealt with by sampling in the training phase. Using the auto-model function for both target attributes produced significantly less impressive results, especially for the second type of undesired outcomes. For the first type of undesired outcomes the Random Fores The choice is, therefore, made to shift the focus of this analysis to simply 'undesired outcomes', where no distinction is made between the two types of undesired outcomes.

Figure 3: Difference between the distributions of the different types of undesired outcomes.

The next step in this analysis was, as explained earlier, rebuilding the best performing model(s) (according to the results in RapidMiner) in Python, using the same set of attributes and model parameters. This allows one to have more control over the built model and it facilitates reproducibility of the results when one releases his/her source code. The downfall is, however, that there is a dependency on third-party libraries so there is a risk that the results are not in line with what is achieved in RapidMiner, which is of course in itself third-party software. The advantage of using Python and its various scientific computing and machine- and deep-learning libraries is that these libraries are often times open source. One can, therefore, adjust specific functions or other bits and pieces if one is not satisfied with a certain implementation and/or when there are optimization possibilities that are linked to a certain specific problem context and respective computing approach.

Considering the performance metrics that were briefly evaluated in the second last paragraph, the choice was made to reproduce the results of the Random Forest model and the Deep Learning model. The reasoning behind these choices can be summarized as follows: On the one hand, one can argue that a simple white-box model such as decision trees allows for a more intuitive and easy evaluation of the underlying logic of the model, but a quick evaluation of, for example, the

resulting decision tree in Rapid Miner shows that one can not derive clear and easy comprehensible implications from the underlying logic. This is probably mostly due to the vast amount of attributes that were fed to the models and the rather complex web of processes and documents that are encapsulated in the data. The focus was, therefore, purely set on answering the first business question: Is it possible to detect undesired outcomes, preferably before the decision is made for the case in question?

Keeping this choice and goal in mind, the choice was made to reproduce the results of more sophisticated models such as Deep Learning models and Random Forest. The goal is to simply illustrate what is possible using these modeling techniques. If the performance metrics and approach are in line with interests of the parties involved, they can initiate, for example, a project in which a simple (web) application is made for the people involved in the process (decision making) that would allow the people involved to have insights in how likely a case is going to end up being an undesired outcome and act upon the results of these (probabilistic) models. The investment in reproducing the results in Python is, therefore, worthwhile when one keeps this possible use case of the resulting models in mind. It is easier to integrate it into production-level code than, for example, models built in RapidMiner or even R. The last and final advantage is simply a general research principle: reproducibility is very important in the scientific world. The majority of the code that is used in sections *3. Preprocessing and exploration of the event logs* and *4. Undesired Outcomes* is, therefore, publicly available on GitHub: EAGF-BPI2018 repository.

The `sklearn` library was used for the Random Forest model and `TensorFlow` was used to build a Neural Network (i.e. Deep-Learning model). Considering the nature of these models (e.g. `RandomForestRegressor` was used from the `sklearn` regressor collection), the data needed to be processed one more time. In particular, one-hot encoding of categorical data was needed for the models build with `RandomForestRegressor` and `TensorFlow`. In addition, some numeric data needed to be bucketed in some cases (as was also done in RapidMiner). The results (accuracy, precision and recall) for the Random Forest model were in line with the aforementioned results that were acquired in RapidMiner. The results of the model that was built using `TensorFlow` was a bit worse (e.g. accuracy was 2 percent point lower) than the Deep Learning model that was produced by RapidMiner. This can be due to several reasons. As explained earlier, the implementations will differ a lot. In addition, the model that was built using `TensorFlow` has been tweaked after some experimentation and evaluation of TensorFlow estimators . This includes hyper-parameters such as number of epochs, the batch size, learning rate and also the design of the actual network (e.g. number of layers). This is done since RapidMiner fails to deliver an extensive report on all of these parameters and characteristics it used, and how they were chosen.

4.4 Conclusive Remarks and Respective Recommendation

The question that was considered is whether it is possible to detect undesired outcome cases as early as possible. Ideally, this would happen before a decision is made for the case in question, which is indicated by the first occurrence of the activity `Payment application-Application-decide`, where one is allowed to use data of previous year to answer this question. Considering the aforementioned results and discussions in this section, the conclusive answer is that this is indeed possible if one considers the situation just before a decision needs to be made.

In addition, the difference in the performances of the model between the situation in which undesired outcomes in general are considered, compared to specific types of undesired outcomes (type 1 and type 2) is significant enough to recommend to limit the modeling efforts to detecting undesired outcomes in general. Considering the impressive performance of these models, the recommendation would be to exploit models that focus on undesired outcomes in general. As explained earlier, the organization(s) involved can devise a project in which a simple (web) application is made for the people involved in the process (decision making) that would allow the people involved to have insights in how likely a case is going to end up being an undesired outcome and act upon the results of these (probabilistic) models.

5 Risk Assessment: Prediction of Penalties

The second business question that will be covered in this report concerns the prediction of penalties, also called the risk assessment. It could happen that the applicant may not receive the total amount of what has been requested. This can have various reasons. The reported size of the parcel does for example not correspond to the actual size of the parcel. Other reasons can be the violation of predetermined conditions or not satisfying the young farmer condition when one applies as a member of this target group. The reason of a penalty is included through the attribute `penalty_{xxx}`. Some of the penalties are more severe than others. As mentioned in *3. Preprocessing and exploration of the event logs*, a severe penalty is any of the penalties in {B3, B4, B5, B5F, B6, B16, BGK, C16, JLP3, V5}. A part of the applications is selected for an on-site inspection. This is done in two different ways: by risk assessment (`selected_risk`) or random (`selected_random`). The dataset owner expects that the risk assessment shows a relatively large part of the severe penalties by selecting a sample from the total population. However, they still see room for improvement.

5.1 Approach

Various predictive models will be built to come to a sample which includes a higher percentage of applicants which should receive a penalty. Before doing this, the performance of the current risk assessment will be analyzed and the data will be preprocessed further. It is chosen to only focus on predicting whether there occurs a severe penalty, since these penalties deserve the most attention. Furthermore, the percentage of regular penalties lies around 20%, which apparently shows that it is quite common to receive a regular penalty. The prediction task will focus on unique applicants (`applicant`) per year. This means that multiple applications per year from one applicant will be left out of scope. For the preprocessing and prediction tasks, both RapidMiner and Python were used. Just as in the previous section, RapidMiner Auto Model was used to construct predictive models. Predictive modeling will be executed by using three different combinations of predictor and response variables (referred to as tasks):

1. Predicting the presence of a severe penalty in 2016 based on penalty data from 2015 and application data from 2016.
2. Predicting the presence of a severe penalty in 2017 based on penalty data from 2015 and 2016 and application data from 2017.
3. Predicting the presence of a severe penalty in a specific year (2016 or 2017) based on penalty data from the year before (2015 or 2016) and application data from the current year.

It is important to note that application data refers to data concerning events that happened before the remote or on-site inspections for the particular year have started. Other attributes could not be taken into account, since these attributes were not known before inspection has taken place and the application is processed. The difference between the second and the last predictive model is that the first model is based on data from the past two years, while the last model is based only on the data from the year before. By comparing the performances of both models, one can assess whether it would be better to predict based on only last years data, or whether it would be better to include the last two years.

5.2 Current performance

To asses the performance of the predictive models, three performance criterion are used: the precision (What percentage of the selected items will be severely penalized?), the recall (What percentage of the severely penalized items is selected?) and the accuracy (How many applications received a correct prediction?). All can be written in mathematical form in the following way:

$$precision = \frac{true\_positives}{true\_positives + false\_positives}$$

$$recall = \frac{true\_positives}{true\_positives + false\_negatives}$$
$$accuracy = \frac{true\_positives + true\_negatives}{total\_population}$$

First, the performance of the risk assessment is investigated. This is done for both 2016 and 2017. In Table 2, some descriptive statistics concerning the number of applicants selected for additional inspection and the number of severe penalties is given for both 2016 and 2017. During the rest of this chapter, we imply that an applicant is selected for on-site inspection by risk assessment when we refer to `selected_risk`. The other selection methods, `selected_manually` and `selected_random` will be left out-of-scope. From Table 2 it can be concluded that the percentage of severe penalties increased from 1.2% to 2.0% when comparing 2016 to 2017. It is interesting to note that this percentage equaled almost 4.0% in 2015. Furthermore, also the number of regular penalties was twice as high as in 2016 and 2017. From this, it can be concluded that either 1) a change was made in the penalty assessment process or 2) less violations of conditions were made.

Table 2: Descriptives of severe penalties in 2016 and 2017

		2016		2017	
Selected risk	Severe penalty applied	Count	Relative	Count	Relative
False	False	13828	95,0%	13728	94,6%
False	True	120	0,8%	191	1,3%
True	False	551	3,8%	489	3,4%
True	True	51	0,4%	100	0,7%

The performance criterion are calculated using the results in Table 2 and can be found in Table 3. It can be seen that the performance of the risk assessment increased in 2017 compared to 2016. The accuracy is outstanding high in both years. This is due to the low number of severe penalties, which will cause the risk assessment model to almost always predict `FALSE`.

Table 3: Performance of current risk assessment

	Precision	Recall	Accuracy
2016	8,5%	29,8%	95,4%
2017	17,0%	34,4%	95,3%

5.3 More specific preprocessing

The data has already been preprocessed as describe in section *3. Preprocessing and exploration of the event logs*. However, additional preprocessing steps are needed before various predictive models can be built. This includes selecting suitable attributes and splitting the dataset to improve the percentage of severe penalties.

5.3.1 Attribute selection

As described earlier, various attributes cannot be used as predictors because they include information on events which happen after the remote and on-site inspections for the particular year have started. These attributes are `penalty_{}`, `amount_applied{}`, `payment_actual{}`, `penalty_amount{}`, `risk_factor`, `cross_compliance`, `selected_random`, `selected_risk`, `selected_manually` and `rejected`. However, the attributes `penalty_{}` and `penalty_amount{}` will be used as predictors when talking about the values from previous years.

The attributes `Activity`, `activity`, `docid`, `doctype`, `eventid`, `Resource`, `Case ID`, `note` (100% missing) and `subprocess` will not be used since they have activity specific values and can therefore possess multiple values for one application. The booleans `greening` and `basic_payment` are always `TRUE` and for that reason of no value when executing a prediction task.

The regular attributes which are used for the prediction task are `number_parcels`, `area`, `department`, `redistribution`, `small_farmer`, `young_farmer`, `timestamp_year` and `timestamp_month`. Besides, one will use the attributes created in the section *3. Preprocessing and exploration of the event logs*: `num changes penalty amount_xxxx`, `final penalty amount_xxxx`, `num_penalty_reasons_xxxx` and `num_severe_penalties_xxxx`. `xxxx` concerns to the specific year the attribute refers to (either 2015 or 2016). Furthermore, the following attribute will be created:

- `previous_penalties_xxxx` - Boolean attribute which is `TRUE` in case at least one severe penalty was received in year `xxxx` and `FALSE` in case no severe penalty was received.

Before the actual prediction task took place, additional preprocessing such as correcting missing values and including year numbers to the above mentioned attributes took place.

5.3.2 Creating a balanced dataset

As shown in *5.2 Current performance*, in 2017 only 2.0% of the applicants receive a severe penalty. Finding true values in a dataset with a 2/98 `TRUE/FALSE` ratio is an unsuitable prediction task. Because of this, it is important to re-balance the dataset. This will be done such that the `TRUE/FALSE` ratio equals 30/70. All the applicants with `severe penalty applied` equal to `TRUE` are preserved. From the example set with `severe penalty applied` equal to `FALSE`, a random sample of 1.5 times the size of the `TRUE`-set is taken. Both example sets are combined to come to a final set which can be used as input for constructing predictive models. It was verified that the balanced dataset lead to a much better performance. These results will not be reported in detail. The improvements could mainly be noticed when using a Generalized Linear Model, Logistic Regression or Random Forest. When using these methods on the complete dataset, the models tend to predict all values as `FALSE` to achieve an extremely high accuracy. Since we want to prevent such form of overfitting, we will from now on use the balanced dataset.

5.4 Predictive Modeling Results

RapidMiner Auto Model will now be used to create various predictive models. Some attributes are filtered out by this software after analyzing the 4 quality measures. One of these quality measures is Stability: measure how stable or constant this column is. It is equal to the number of rows with the most frequent non-missing value divided by the total number of data rows with non-missing values. There are 4 attributes with a stability of almost 100%: `redistribution`, `small_farmer`, `young_farmer` and `timestamp_year`. Since one strives for a stability as low as possible, these attributes are removed from the list of input variables.

Table 4: Predictive modeling performance Task 1 and Task 2

	Task 1			Task 2		
Year predicted		2016			2017	
Penalty data from		2015			2016	
Application data from		2016			2017	
	Precision	Recall	Accuracy	Precision	Recall	Accuracy
Naïve Bayes	65,0%	38,2%	67,1%	54,1%	34,5%	62,1%
Generalized Linear Model	66,7%	23,5%	64,7%	66,7%	24,1%	64,8%
Logistic Regression	54,5%	35,3%	62,4%	51,3%	34,5%	60,7%
Deep Learning	55,9%	55,9%	64,7%	47,1%	56,9%	57,2%
Decision Tree	75,0%	35,3%	69,4%	64,3%	15,5%	62,8%
Random Forest	71,4%	29,4%	67,1%	78,6%	19,0%	65,5%
Gradient Boosted Trees	46,2%	35,3%	57,6%	42,5%	63,8%	51,0%

In Table 4 and Table 5, the performance measures of various predictive models can be found for each prediction task. The accuracy is almost always equal to 60%. This is much lower than the performance of the current risk assessment since the predictive models were built based on a balanced instead of the complete dataset. The accuracy will therefore not be used to assess the performance of the new model.

Table 5: Predictive modeling performance Task 3

	Task 3		
Year predicted Penalty data from Application data from		2016 and 2017 previous year current year	
	Precision	Recall	Accuracy
Naïve Bayes	65,3%	34,8%	66,7%
Generalized Linear Model	63,6%	30,4%	65,4%
Logistic Regression	59,7%	46,7%	66,2%
Deep Learning	56,6%	69,6%	66,7%
Decision Tree	72,7%	17,4%	64,5%
Random Forest	67,9%	20,7%	64,5%
Gradient Boosted Trees	57,6%	57,6%	66,2%

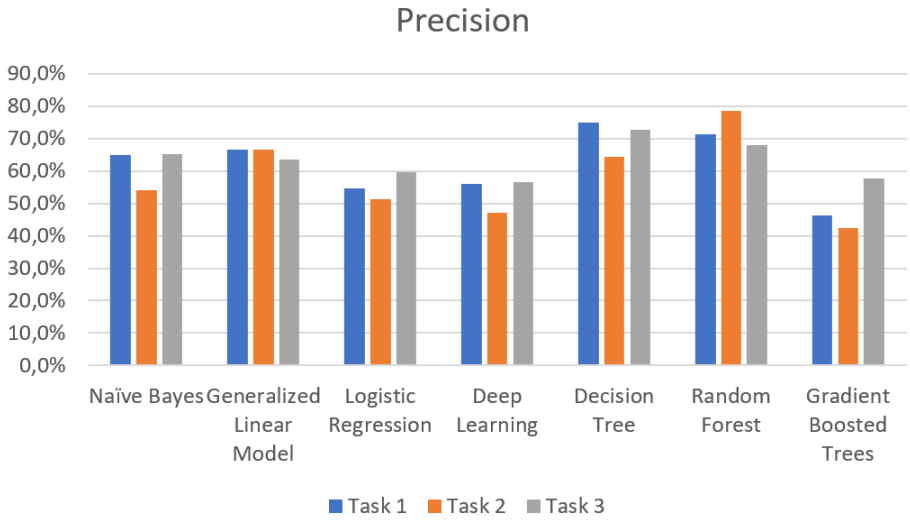


Figure 4: Overview of precision performance of various models

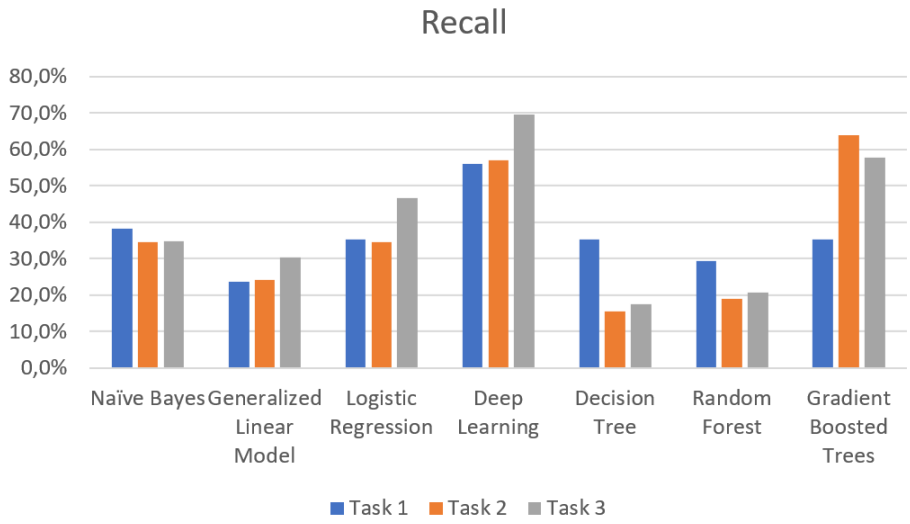


Figure 5: Overview of recall performance of various models

In Figure 4 and Figure 5, barcharts of the precision and recall performance can be found. This enables us to easily compare the performance of various predictive models. Since our goal was to increase the recall of the risk assessment, we will first focus on Figure 5. The recall for the original risk assessment in 2016 was 29.8% and 34.4% in 2017. It can be seen that Naïve Bayes, Logistic Regression, Deep Learning and Gradient Boosted Trees model exceed these values for all prediction tasks. Of these models, Deep Learning and Gradient Boosted Trees have the highest performance. The precision for the original risk assessment in 2016 was 8.5% and 17.0% in 2017. In Figure 4 is can be seen that all predictive models exceed the original values. For the Deep Learning Model and the Gradient Boosted Trees Model, the performance values for prediction Task 3 are almost always highest compared to Task 1 and Task 2. We can therefore conclude that improving the prediction model with each year's new data gives the best overall performance. Since the precision of the Deep Learning model is higher than the precision of the Gradient Boosted Trees model for all prediction tasks and since the Deep Learning models tend to have the most consistent recall performance, one will have a look at the Deep Learning model with more detail. This will be done for prediction task 3, since this gives the best overall performance as described above.

The Deep Learning Model created consists of four layers and does not use dropout. The model completes ten epochs, with a batch size equal to the number of training examples (925). In Table 6 the confusion matrix of the Deep Learning Model can be found. Since the dataset was balanced during preprocessing, it can be seen that the number of examples used to train en test the models is very low. The dataset which was used to create the prediction task 3 models consists of 1156 records. This is data for both 2016 and 2017. To improve the reliability of the models, it would be better to create predictive models based on more examples.

		Predicted Value		
		True	False	Precision
Actual Value	True	64	28	56.6%
	False	49	90	76.3%
	Recall	69.6%	64.8%	

Table 6: The confusion matrix that resulted from the Deep Learning Model.

5.5 Conclusions and recommendations

In this section, various predictive models were built for predicting whether a severe penalty is applied to a specific applicant. It was shown that the recall performance can be increased from 34.4% to 69.6% by enriching the dataset with more attributes and building a Deep Learning Model based on these input attributes. Initially, the sample size of the risk assessment was about 4.0%. Since the recall almost doubled and the number of severe penalties in 2016 and 2017 is lower than the number of severe penalties in 2017, a lower sample size suffices when using the new predictive model. This sample size will even be smaller than 1.0%. The attributes `num changes penalty amount_xxxx`, `final penalty amount_xxxx`, `num_penalty_reasons_xxxx` and `num_severe_penalties_xxxx` were added to enrich the dataset. However, the predictive model can probably still be improved by making several adjustments. The analysis in the next section can be used to enrich the model even more. One could for example include attributes concerning events which were or were not executed until the risk assessment is executed. Furthermore, one could include the throughput time of specific events as attributes.

As described earlier, it would be better to increase the number of examples in the dataset. The current balanced dataset namely creates a model based on 925 training records. Furthermore, the balanced dataset could be made more representative for the complete dataset when one uses clustering instead of random sampling. When using clustering, sampling the `Severe penalty applied = FALSE` records will happen in a more substantiated way. By first training and testing on the balanced dataset, one can create a first version of the predictive model. The next step is testing on the complete dataset, which will probably lead to a lower recall performance.

6 Differences between Departments and across Years

The dataset comprises cases from four different departments, labeled 4e, 6b, 4d, and e7 across three years (2015, 2016, 2017). Making subsets for each combination and comparing these may prove useful. For one, as exploratory analysis of application attributes may show interesting differences. Secondly, because comparing performance measurements may reveal best practices. To do this, Disco and ProM are used. Disco is used to make and compare process models. ProM helps obtain insights in the throughput time of process models. Additionally, Python is used where Disco and ProM fall short, such as for calculations of certain averages.
The remainder of this section is structured as follows. First, several attributes such as the applied amount are compared across years and between departments. Secondly, the process models themselves are shown and compared. Then, an attempt is made to measure process performance and find best practices.

6.1 Attributes

As it is important to investigate all possible changes between years and departments the attributes were compared as well. The following attributes were examined; the average amount applied per department, the type of applicant, the area size and number of parcels. These will be threated in this order. Other attributes were also researched but showed no (significant) differences.

The average amount of money for which farmers applied will be discussed first (Figure 6. The first finding was that the average amount increased a bit every year for each department. There was no real difference in increase between the different departments, so this is just a difference between years. Furthermore, every year less changes were made during the process for the applied amount. This is remarkable, because this could imply that the process did improve over the years. The changes that were made did match with the payment_actual en penalty_amount and this means that there were no big mistakes or typos made with filling in the numbers. An overview of the specific results can be found in 8.1[Appendix A] .

Year / Department	2015	2016	2017	All
4e	19776	19775	19774	19768
6b	29824	29902	30030	29899
d4	27597	27513	28826	27835
e7	22130	22317	22326	22244
All	24093	24152	24365	24175

Figure 6: Average initial application amount

Furthermore it is interesting to see if the application process differs when the applicant is of a different type, such as a young farmer. This proved to not be the case. The last two attributes that are examined are average area size and the average number of parcels per year and department (Figure 7 and Figure 8 respectively).

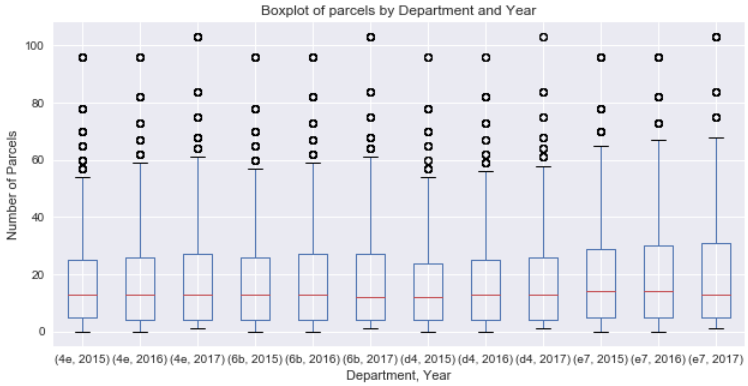


Figure 7: Boxplot of number of parcels and department

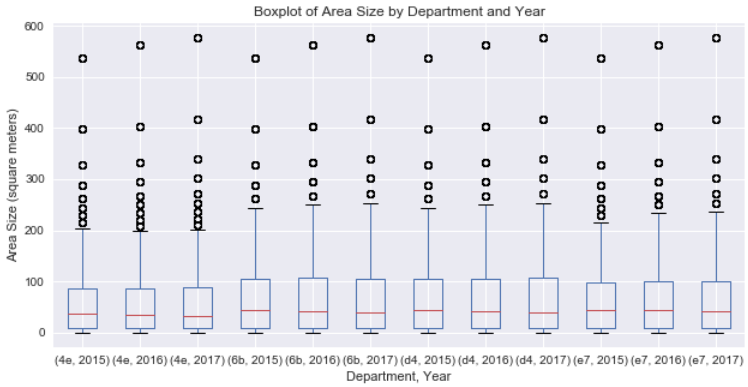


Figure 8: Boxplot of area size and department

When we take the different years into account. We find that the average number of parcels per department slightly tend to increase while the average area size stays almost the same per department, which can be logically expected. The area size and number of parcels are expected to be highly correlated. Nonetheless, Figure 9 below shows that while there is a correlation, the area size cannot be completely predicted from the number of parcels, or the other way around.

6.2 Process differences

The data was loaded in Disco and attribute filters were used to obtain process models for the individual years and departments. An advantage of Disco is that it allows for visual inspection of the results, which makes finding differences in the main processes easy. Unfortunately, large deviations were not found. Some subtle differences are explained below.

As mentioned in section 2 the 'Parcel' document was replaced by the 'Geo Parcel' document. The difference in the Disco models are showed in figure 10. As you can see after an employee is finished with editing the 'Parcel' document it can go to the initializing of the control summary or the initializing of the reference alignment, while the 'Geo Parcel' document of 2016 follows a straight line in this part of the process. The reason for this could be that in 2015 it happened a lot that a loop was needed for a case. As this is not the case in 2016 this might have saved extra work and paperwork (by means the use of less documents). However when we take a look on the time it takes, the handling of both documents take about the same time.

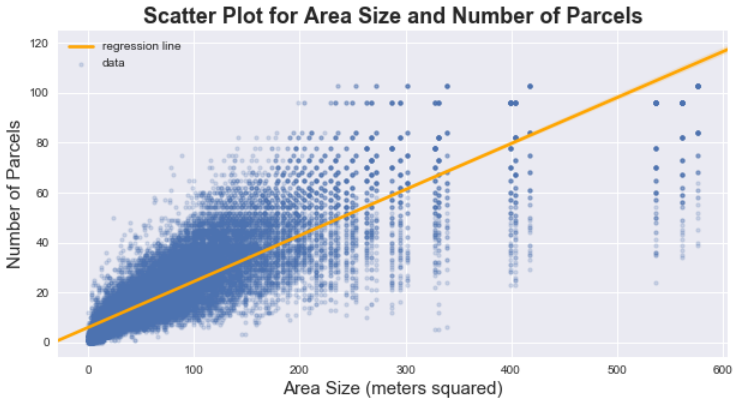
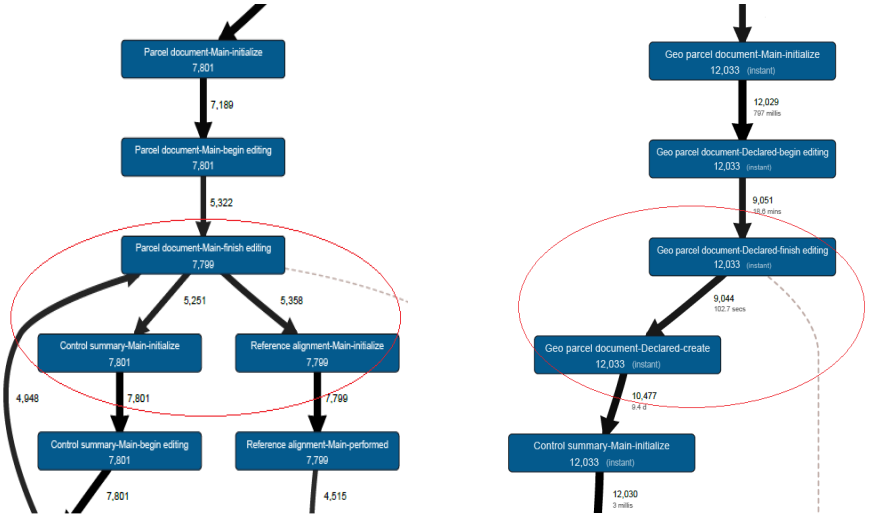


Figure 9: The correlation between area size and number of parcels

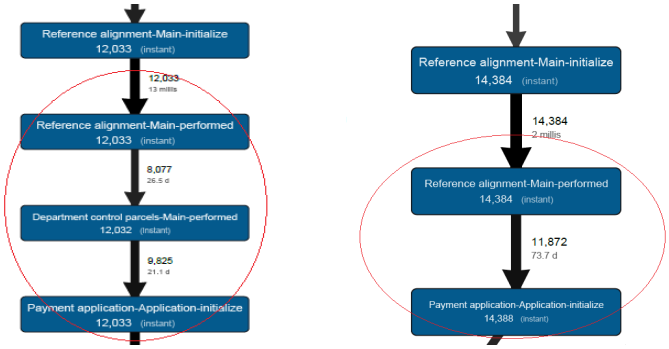


(a) Situation 2015 *(b)* Situation 2016

Figure 10: Differences between process changes 2015 and 2016

In Section 2 was mentioned that the 'Department Control Parcel' was adapted in the year 2017. This means that this document did not exist in 2017. This is shown in figure 11, where one step is left out in 2017. When we look at savings of time it is interesting to see that there was the reference alignment in but years happened around the same time but that the average time it takes for a case in 2016 to go from reference alignment to go through a department controlling the parcels and initialize the payment application takes around 48 days, while it is in 2017 around 74 days (and this is without the department controlling the parcels). We see that there are a lot of standard starting points so it is not possible for us to see that the differences between 48 days and 74 days is of importance for the EAGF. However, this adaption might at least result in less paperwork.

Now we will take a look at the number of events per case which we calculated. Disco is used to obtain both the average number of events per case, as well as a distribution of the number of events per case. In general, this distribution resembles a sharply dropping exponential distribution, with around 90% of the cases falling in the lowest 10% of events per case. Figure 12 shows the average number of events per case. The year 2015 has been filtered out for this comparison, as 2015 is quite dissimilar from 2016 and 2017 and pollutes the data significantly. The activities are the number of unique activities. It shows that department e7 has at least 7 activities which the other departments do not have. Careful examination shows that in total 153 unique activities happened in 2016



(a) Situation 2016 *(b)* Situation 2017

Figure 11: Differences between process changes 2016 and 2017

and 2017, revealing even more differences between departments. Nonetheless, the main activities are the same, and those that do differ are activities such as *Geo parcel document-Declared-remove document* which occurs in five cases from the e7 department. Since similar low-frequency activities which are unique to departments occur, it seems unlikely that this is purely due to chance. Instead, it is likely that departments do have subtle differences in their process.

	4e	6b	d4	e7
Cases	9059	7463	3814	8723
Events (x1000)	478	422	248	468
Activities	135	134	134	142
Events per case	52,7	56,6	65,1	53,7

Figure 12: Average Number of Events per Case

The number of events per case seems rather high, and shows quite some differences across departments. Careful inspection of the data reveals that one activity, *inspection-on-site-save* repeats itself often. In one case specifically, this event is repeated 2713 times. This event may distort the results. Therefore, it is removed and the new averages are calculated. This is shown in Figure 13. Main activities and repetitions are checked. Unfortunately, the differences could not be attributed to any specific activity. Further research might be conducted into where this difference stems from.

	4e	6b	d4	e7
Cases	9059	7463	3814	8723
Events per case	50,8	54,2	59,7	50,2

Figure 13: Number of Events per Case

6.3 Performance

Performance can be measured in many ways. Earlier in this report, a few undesired outcomes, as well as penalties, have already been discussed. Furthermore the events per case were calculated per department. Nevertheless, measures such as throughput time and number of penalties are still useful to investigate.

Besides the earlier calculations of events per case the throughput time is another measure which may be used for the performance. Throughput time itself cannot always be used as a performance measure, since larger throughput times may not necessarily be bad. Nonetheless, differences can be reported and investigated more thoroughly. The actual calculations are done in Python, by using the datetime module and subtracting the last activity's timestamp from the first.

A starting point is the difference across years. A first observation shows that there are significant differences (see Figure 14). The averages differ quite a bit, as well as the other measurements. Upon further inspection, however, most of these seem insignificant. Firstly, the maximums, especially the one for 2015, might very well be manual mistakes in entering timestamps. This is supported by manual checking of these cases, which reveals that cases which such long times often have end activities which are unlikely to be the actual end activities. This also seems to be the case for the minimum throughput time in 2017. Next, while the average and median show difference, these can be largely attributed to a single date in which most of the payments happen; for each year, there is a single date on which most payments are made. For the 2015 cases, this date is in the middle of February. Instead, for 2016 and 2017 cases, this occurs in the beginning of January. This largely explains the differences in throughput times. The standard deviation can be explained as deviations in starting times of cases. It can be concluded that there is an insignificant difference in throughput time across year. The same can be said between departments, in which the total difference between lowest throughput time (department 6b) and highest throughput time (department e7) is only two days.

Year (started in)	2015	2016	2017
Average	291 days	243 days	251 days
Standard Deviation	11 days	17 days	13 days
Minimum	217 days	198 days	120 days
Maximum	621 days	394 days	284 days
Median	289 days	242 days	248 days

Figure 14: Throughput Measures per Year

Lastly, the departments and years were compared on number of penalties. The same distinction is made between penalties and severe penalties. Both of these show interesting differences (Figure 15). More specifically, cases in departments 6b and d4 have the highest average number of penalties per case (almost 40% of the cases are penalized). Furthermore, in 2015 the relative number of penalties is almost thrice as high as in 2016 and 2017.

	2015	2016	2017
Number of penalties	9388	3368	2967
Number of cases	14749	14552	14507
Number of penalties per case	0.637	0.231	0.204

(a) Per Year

	4e	6b	d4	e7
Number of penalties	4649	4438	2252	4384
Number of cases	13639	11256	5743	13170
Number of penalties per case	0.341	0.394	0.392	0.333

(b) Per Department

Figure 15: Number of Penalties across Years and between Departments

The severe penalties show the same trend, with departments 6b and d4 having slightly higher numbers of penalties per case (around 5% versus around 3%) and 2015 having over four times as many penalties per case.

One possible explanation for the differences in number of penalties applied could be the number of checks that are performed by each department. For d4, this might explain it partly, as they select significantly more cases to check (15% vs 10% for the other departments). Nonetheless, the difference is not fully explained by this alone. Further inspection reveals that not only do checks not correlate with penalties, but many penalties are given without checks being performed and vice versa. This is, in fact, not inherently bad. Instead, it shows that for some reason, cases in department 6b (and to some extent d4) are more susceptible to receiving penalties. More insight and knowledge of the underlying processes and the division of departments is needed to find a fitting explanation for this. However, possibilities might include: (1) that these departments are more strict or penalize faster, (2) that applications in these departments are more prone to being penalized, or (3) that applicants in these departments are more prone to being penalized.

7 Discussion

While the performances of (most of) the models for the detection of undesired outcomes were considered to be adequate and in some cases impressive, as described in *4. Undesired Outcomes*, there is room for improvement and some things that can and should be reevaluated. The choice was made to not go back too much in the case life-cycle when preprocessing the data for the undesired outcomes model. One can, however, imagine that it would be better to assess whether going back even further in the case life-cycle without degrading the performance metrics too much. This is, however, not explored considering the time aspect of the research project.

In *Risk Assessment: Prediction of Penalties*, various predictive models were built for predicting whether a severe penalty is applied to a specific applicant. Almost all models showed a significant improvement compared to the original risk assessment. It was shown that the recall performance can be increased from 34.4% to 69.6% by enriching the dataset with more attributes and building a Deep Learning Model based on these input attributes. Furthermore, it is advised to built an initial predictive model and improve this model each year with new data.

Some interesting findings related to question 3 and 4 is to keep into mind were that the changes made during the process for applied amount decreased every year. This means that less work has to be done and if this is a trend it can be interesting to keep track of it to make sure it will not increase again. That the number of parcels per department slightly increase could mean that the parcels that exist become more divided between farmers which could result in more payment applications, so this is a trend to keep into account as well.

Changes within the process between years were found in the dataset, but as we do not have intern information about where the relocation of work happened, is it hard to get more insights without making too many assumptions. It might for example be interesting for EAGF to see how time is spend now in contrast to last year between the reference alignment and the initializing of the payment application. Besides, as the average events per case and the count of penalties show differences between departments, while there could be different reasons for this as mentioned earlier, it might be interesting to take a look at this with more intern knowledge of the process. Therefore, one of our recommendations is to have an analyst go through the dataset for these differences with some more intern business contexts.

References

[de] data experts. profile c/s. https://www.data-experts.de/produktloesungen/profil/.

[Gan04] Jack Ganssle. Chapter 14 - {IEEE} 754 floating point numbers. In Jack Ganssle, editor, *The Firmware Handbook*, Embedded Technology, pages 203 – 205. Newnes, Burlington, 2004.

[SDC+17] T. Swearingen, W. Drevo, B. Cyphers, A. Cuesta-Infante, A. Ross, and K. Veeramachaneni. Atm: A distributed, collaborative, scalable system for automated machine learning. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 151–162, Dec 2017.

[VDB18] B.F. (Boudewijn) Van Dongen and F. (Florian) Borchert. Bpi challenge 2018, 2018.

8 Appendices

8.1 Appendix A: Differences across Years and between Department

This appendix contains all results which were not included in the attribute subsection of differences between departments and across years. It is mostly focused on the application, payment, and penalty amounts. Each number (1, 2, 3) shows the number of times a correction was made. The number of cases over which the average was computed is also included. This shows that corrections do sometimes occur, however, two or three corrections hardly ever occur. Moreover, some statistics on the type of farmer are included as well.

Year / Department	2015	2016	2017	All
4e	44130	49276	15688	46192
6b	80228	86502	-	83025
d4	72967	76290	-	74130
e7	44521	58383	98711	50033
All	58790	67855	89486	62484

(a) Average amount

Year / Department	2015	2016	2017	All
4e	425	304	2	731
6b	455	366	0	821
d4	310	167	0	477
e7	555	273	16	844
All	1745	1110	18	2873

(b) Situation 2016

Figure 1: Average applied amount after one correction

Year / Department	2015	2016	2017	All
4e	81167	83761	-	82311
6b	93963	142695	-	105659
d4	90529	104719	-	95259
e7	71126	58530	-	66928
All	85734	96778	-	89346

(a) Average amount

Year / Department	2015	2016	2017	All
4e	19	15	0	34
6b	38	12	0	50
d4	26	13	0	39
e7	24	12	0	36
All	107	52	0	159

(b) Situation 2016

Figure 2: Average applied amount after two corrections

Year / Department	2015	2016	2017	All
4e	-	49107	-	49107
6b	87523	-	-	87523
d4	368979	42167	-	257737
e7	69756	42167	-	55961
All	165401	42176	-	128433

(a) Average amount

Year / Department	2015	2016	2017	All
4e	0	1	0	1
6b	4	0	0	4
d4	2	1	0	3
e7	1	1	0	2
All	7	3	0	10

(b) Situation 2016

Figure 3: Average applied amount after three corrections

Next, the type of farmers is shown.

Attribute	# True	# False
Basic payment	37462	0
Redistribution	37392	70
Greening	37462	0
Small farmer	1757	35705
Young farmer	3003	34459

Figure 4: Number of special farmers

	Redistribution	Small farmer	Young Farmer
4e	11633	567	953
6b	9612	464	699
d4	4916	275	366
e7	11231	451	985

Figure 5: Number of special farmers per department